expressly or inherently described, in a single prior art reference," citing Verdegaal Bros. v. Union Oil Co. of California, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987). Moreover, MPEP §2131 indicates that the cited reference must show the "identical invention . . . in as complete detail as is contained in the . . . claim," citing Richardson v. Suzuki Motor Co., 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989). For the reasons identified below, Applicants submit that the Examiner has failed to establish anticipation of claims 1-4, 6-9 and 14-18 by Edwards.

Independent claim 1, by way of example, is directed to a method for use in providing improved fault tolerance in a computing system comprising at least one computing machine. The method includes the steps of executing a control program in conjunction with a fault tolerance software system running on the at least one computing machine, and initiating via the control program a test script program which sends one or more requests to a monitored program, wherein the test script program processes corresponding responses to the one or more requests, and generates at least one return value utilizable by the control program to indicate a failure condition in the monitored program.

A "fault tolerance software system" as described in the specification is a system which includes a failure detection component and a failure recovery component. See the specification at, for example, page 1, lines 19-27:

> The above-noted conventional fault tolerance systems typically include a failure detection component and a failure recovery component. The failure detection component determines if a monitored application, process or other program has terminated, aborted or otherwise failed. For example, in the above-noted SwiFT system, a monitoring process referred to as *watchd* serves as the failure detection component. The recovery component initiates recovery actions in the event that a failure is detected by the failure detection component. A given recovery action may involve restarting the program on the same machine or another machine. As is well known, a program may be restarted from its initial starting point or via rollback to a designated checkpoint subsequent to its initial starting point.

An illustrative embodiment of the claimed invention is implemented in conjunction with fault tolerance middleware 120 that runs on a server machine 104 of a computing system 100. As shown in FIG. 3, the fault tolerance middleware 120 is arranged between operating system 122 and a monitored application 124 on the server machine 104.

FIG. 4A illustrates an example of the interaction between fault tolerance middleware 120 and a particular monitored application, namely, a server program 140. In this example, the fault tolerance middleware 120 comprises a failure detection component 130 and a recovery component 132. The failure detection component 130 has associated therewith a control thread 134 and a test script 136. The control thread 134 is generated by the failure detection component 130 and periodically invokes the test script 136. The test script sends one or more client requests to the server program 140 and then determines if the correct response or responses have been received from the server program. The test script then sends a return value to the control thread to indicate whether all responses were received correctly. If any response is not received correctly, then the control thread instructs the recovery component 132 to initiate appropriate recovery actions.

This illustrative embodiment is an example of what is referred to in the specification as "periodic external self-test." The term "external" indicates that the test script 136 operates independently of any conventional failure detection provided by the fault tolerance middleware. See the specification at, for example, page 5, lines 25-29, and page 7, lines 6-24.

An arrangement of this type is particularly advantageous in that the test scripts can be written independently of the original fault tolerance middleware, and do not require any modification of the monitored program, either in terms of recompilation or in terms of the addition of static or dynamic libraries. See the specification at, for example, page 15, lines 10-16. Also, it can correct more failures than conventional arrangements. For example, an embodiment of the invention involving a modified *watchd* process significantly outperforms the conventional *watchd* process of the above-noted SwiFT fault tolerance software system. See Table 1 on page 13, and the text at page 12, line 14, to page 14, line 27.

Applicants submit that the Edwards reference fails to meet the limitations of claim 1, and fails to provide its associated advantages as outlined above.

Edwards describes a computer software development tool, namely, a debugger. At column 1, lines 7-11, Edwards identifies the field of his invention as follows, with emphasis supplied:

> The present invention relates generally to computer software development tools and more particularly to a method for debugging a Java application that includes native method dynamic load libraries (e.g., C or C++ code).

Those skilled in the art of computer science will readily appreciate that a debugger or other computer software development tool is distinct from a fault tolerance software system. As noted above, a fault tolerance software system includes not only fault detection capability, but also fault recovery capability. Hence the term "fault tolerance." A debugger such as that disclosed in Edwards is not a fault tolerance software system, in that it does not exhibit "fault tolerance." Instead, it is used to discover faults in a software program that is under development, such that the program designer can take appropriate actions, such as altering the program, in order to correct the faults. The debugger itself is not configured to provide recovery from any detected faults. That is the job of the program designer in the Edwards arrangement. This is apparent from, for example, column 4, lines 9-14, which provides as follows, with emphasis supplied:

> As is well-known, representative debug "events" comprise breakpoint events, single step events, a load events (which may include dll events) and/or fault events. The function of a debugger is to identify such event(s) in the target application (i.e. the application being debugged) and report those events back to the user.

Again, the user, and not the debugger, is responsible for correcting faults, and thus the Edwards debugger is not a "fault tolerance software system" as claimed.

The Examiner in formulating the §102(b) rejection argues that the teachings in the Edwards reference at column 3, lines 62-67, and column 4, lines 2-14, anticipate each and every limitation of claim 1. Applicants respectfully disagree. The relied-upon portions of Edwards relate to debugger 19 as shown in FIG. 2. As Applicants described above, this debugger does not comprise a "fault tolerance software system."

Applicants also wish to point out that the §102(b) rejection is deficient on other grounds. For example, even if one were to assume for purposes of argument that the Edwards debugger 19 constitutes a fault tolerance software system, Edwards still fails to anticipate the claimed

arrangement. Claim 1 calls for three distinct elements, namely, a control program, a fault tolerance software system, and a test script program. Improved fault tolerance, beyond that otherwise associated with the fault tolerance software system, is provided through use of the control program and test script program, which provide a type of "periodic external self-test" as described previously. The claim specifies that the control program executes in conjunction with the fault tolerance software system, and that the test script program is initiated via the control program. The Examiner apparently argues, with reference to FIG. 2 of Edwards, that the debug engine 22 corresponds to the claimed control program, that the graphical user interface (GUI) 21 corresponds to the claimed fault tolerance software system, and that the probe 24 corresponds to the claimed test script program. However, the GUI 21 cannot itself be viewed as a fault tolerance software system. Instead, each of the elements 21, 22 and 24 is identified in Edwards as being an element of the debugger 19. Also, Edwards in column 4, lines 2-4, indicates that the debug engine 22 "performs all of the debugging work." Finally, GUI 21 and debug engine 22 are related as "front end" and "back end" of debugger 19. Thus, even if one were, for purposes of argument, to view the debugger 19 as a fault tolerant software system, its collective elements fail to meet the limitations associated with provision of improved fault tolerance via the claimed control program and test script program.

Accordingly, Edwards fails to disclose an arrangement for providing improved fault tolerance, involving the execution of a control program in conjunction with a fault tolerance software system running on at least one computing machine of a computing system, and the initiation via the control program of a test script program which operates to provide improved fault tolerance for the system as claimed.

Since Edwards fails to meet at least one limitation of independent claim 1, and also fails to provide its associated advantages in terms of improved fault tolerance in a computing system, that claim is not anticipated by Edwards.

Independent claims 17 and 18 include limitations similar to those of claim 1, and are believed allowable for substantially the same reasons that claim 1 is believed allowable.

Dependent claims 2-16 are believed allowable for at least the reasons given above with regard to independent claim 1, and are also believed to define separately-patentable subject matter over Edwards, as indicated below.

With regard to claim 3, this claim specifies that the control program comprises a control thread of a failure detection process associated with a failure detection component of the fault tolerance software system. The Examiner argues that such an arrangement is "inherent" in Edwards because the debug engine 22 "is a Java element." Applicants respectfully disagree. First, debug engine 22 is not described in Edwards as a Java element. Instead, Edwards at column 3, lines 52-61, simply indicates that debug engine 22 "is preferably implemented in software," without specifying that the debug engine is written in any single language. Second, even if one assumes that debug engine 22 is "a Java element," that does not mean that it necessarily comprises a control thread of a failure detection process associated with a failure detection component of the fault tolerance software system.

With regard to claim 4, this claim specifies that the control program comprises a thread of a failure detection process and the test script program comprises a process separate from the failure detection process. The Examiner apparently argues that the debug engine 22 of Edwards corresponds to the claimed control program, and that probe 24 corresponds to the claimed test script program. However, the relied-upon portions of Edwards, namely, column 7, lines 58-65, and column 8, lines 42-51, fail to provide any teaching or suggestion to the effect that debug engine 22 comprises a thread of a failure detection process and probe 24 comprises a process separate from the failure detection process. To the contrary, debug engine 22 is simply described in Edwards as performing "all the debugging work."

With regard to claim 5, this claim specifies that the control program comprises a thread of a failure detection process and the test script program comprises a thread of the same failure detection process. The Examiner argues that such an arrangement is obvious in view of Edwards, but there is no teaching or suggestion in Edwards to the effect that debug engine 22 and probe 24 are configured in the particular manner claimed.

With regard to claim 6, this claim specifies that the test script program is implemented in an object-oriented programming language such that one or more components of the test script program comprise a base class from which one or more other components of the test script program are generatable for use with the monitored program. The Examiner relies on the teachings in column 6, line 61, though column 7, line 16, of Edwards. However, these portions fail to meet the limitation in question. The Examiner argues that the probe 24 of Edwards corresponds to the claimed test

script program, but the relied-upon portions of Edwards fail to provide any disclosure regarding generation of one or more components of probe 24 from other components of probe 24 comprising a base class.

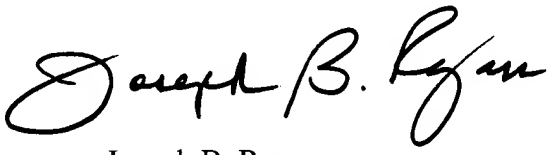
Similarly, the specific test script program component structures identified in claims 8 and 9 are not suggested, much less anticipated, by Edwards.

With regard to claims 14, 15 and 16, the Examiner argues that the test script program corresponds to the probe 24 of Edwards, but relies on column 9, lines 24-39, as allegedly showing the test script program limitations in question. However, the relied-upon portion of Edwards relates primarily to the language of the target program, and fails to disclose that probe 24 comprises an interpreted script, a native executable or a byte code.

In view of the above, Applicants believe that claims 1-18 are in condition for allowance, and respectfully request withdrawal of the §102(b) and §103(a) rejections.

As indicated previously, a Notice of Appeal is submitted concurrently herewith.

Respectfully submitted,



Date: February 4, 2005

Joseph B. Ryan
Attorney for Applicant(s)
Reg. No. 37,922
Ryan, Mason & Lewis, LLP
90 Forest Avenue
Locust Valley, NY 11560
(516) 759-7517

Enclosure(s): Notice of Appeal